The Case Statement

The case statement compares an expression to a series of cases and executes the statement or statement group associated with the first matching case:

- case statement supports single or multiple statements.
- Group multiple statements using begin and end keywords.

Syntax of a case statement look as shown below.
case ()
< case1 > : < statement >
< case2 > : < statement >
.....
default : < statement >
endcase

Example- case

```
module mux (a,b,c,d,sel,y);
input a, b, c, d;
input [1:0] sel;
output y;

reg y;

always @ (a or b or c or d or sel)
case (sel)
  0 : y = a;
  1 : y = b;
  2 : y = c;
  3 : y = d;
  default : $display("Error in SEL");
endcase

endmodule
```

Example- case without default

```
module mux_without_default (a,b,c,d,sel,y);
input a, b, c, d;
input [1:0] sel;
output y;

reg y;

always @ (a or b or c or d or sel)
case (sel)
  0 : y = a;
  1 : y = b;
  2 : y = c;
  3 : y = d;
  2'bxx,2'bx0,2'bx1,2'b0x,2'b1x,
  2'bzz,2'bz0,2'bz1,2'b0z,2'b1z : $display("Error in SEL");
endcase

endmodule
```

The example above shows how to specify multiple case items as a single case item.

The Verilog case statement does an identity comparison (like the === operator); one can use the case statement to check for logic x and z values as shown in the example below.

✦ Example- casez

```
module casez_example();
reg [3:0] opcode;
reg [1:0] a,b,c;
reg [1:0] out;

always @ (opcode or a or b or c)
casez(opcode)
  4'b1zzx : begin // Don't care about lower 2:1 bit, bit 0 match with x
              out = a;
              $display("@%0dns 4'b1zzx is selected, opcode %b",$time,opcode);
            end
  4'b01?? : begin
              out = b; // bit 1:0 is don't care
              $display("@%0dns 4'b01?? is selected, opcode %b",$time,opcode);
            end
  4'b001? : begin  // bit 0 is don't care
              out = c;
              $display("@%0dns 4'b001? is selected, opcode %b",$time,opcode);
            end
  default : begin
              $display("@%0dns default is selected, opcode %b",$time,opcode);
            end
endcase

// Testbench code goes here
always #2  a = $random;
always #2  b = $random;
always #2  c = $random;

initial begin
  opcode = 0;
  #2  opcode = 4'b101x;
  #2  opcode = 4'b0101;
  #2  opcode = 4'b0010;
  #2  opcode = 4'b0000;
  #2  $finish;
end

endmodule
```

✦ Example- case with x and z

```
module case_xz(enable);
input enable;

always @ (enable)
case(enable)
  1'bz : $display ("enable is floating");
  1'bx : $display ("enable is unknown");
  default : $display ("enable is %b",enable);
endcase

endmodule
```

You could download file case_xz.v here

The casez and casex statement

Special versions of the case statement allow the x ad z logic values to be used as "don't care":

- casez : Treats z as don't care.
- casex : Treats x and z as don't care.

✦ Example- casex

```verilog
module casex_example();
reg [3:0] opcode;
reg [1:0] a,b,c;
reg [1:0] out;

always @ (opcode or a or b or c)
casex(opcode)
  4'b1zzx : begin // Don't care  2:0 bits
              out = a;
              $display("@%0dns 4'b1zzx is selected, opcode %b",$time,opcode);
            end
  4'b01?? : begin // bit 1:0 is don't care
              out = b;
              $display("@%0dns 4'b01?? is selected, opcode %b",$time,opcode);
            end
  4'b001? : begin // bit 0 is don't care
              out = c;
              $display("@%0dns 4'b001? is selected, opcode %b",$time,opcode);
            end
  default : begin
              $display("@%0dns default is selected, opcode %b",$time,opcode);
            end
endcase

// Testbench code goes here
always #2 a = $random;
always #2 b = $random;
always #2 c = $random;

initial begin
  opcode = 0;
  #2 opcode = 4'b101x;
  #2 opcode = 4'b0101;
  #2 opcode = 4'b0010;
  #2 opcode = 4'b0000;
  #2 $finish;
end

endmodule
```

✦ Example- Comparing case, casex, casez

```verilog
module case_compare;

reg sel;

initial begin
  #1  $display ("\n    Driving 0");
  sel = 0;
  #1  $display ("\n    Driving 1");
  sel = 1;
  #1  $display ("\n    Driving x");
  sel = 1'bx;
  #1  $display ("\n    Driving z");
  sel = 1'bz;
  #1  $finish;
end

always @ (sel)
case (sel)
  1'b0 : $display("Normal : Logic 0 on sel");
  1'b1 : $display("Normal : Logic 1 on sel");
  1'bx : $display("Normal : Logic x on sel");
  1'bz : $display("Normal : Logic z on sel");
endcase

always @ (sel)
casex (sel)
  1'b0 : $display("CASEX  : Logic 0 on sel");
  1'b1 : $display("CASEX  : Logic 1 on sel");
  1'bx : $display("CASEX  : Logic x on sel");
  1'bz : $display("CASEX  : Logic z on sel");
endcase

always @ (sel)
casez (sel)
  1'b0 : $display("CASEZ  : Logic 0 on sel");
  1'b1 : $display("CASEZ  : Logic 1 on sel");
  1'bx : $display("CASEZ  : Logic x on sel");
  1'bz : $display("CASEZ  : Logic z on sel");
endcase

endmodule
```

Case structure – the case structure has a slightly different syntax than its counterpart in C++ in that no break statement is necessary and the word switch is not used. A typical case structure might look like this:

```
case (t)
      0: y = w[0];
      1: y = w[1];
      …
      7: y = w[n];
      default: y = x;
endcase
```

In this example each alternative in the case statement takes up only one line. Multiline blocks can be used with begin and end statements.

There are two other "flavors" of the case statement in Verilog. These are called casex and casez. The original case statement checks the alternatives for an exact match of 0, 1, x, or z. The casex statement treats both x and z as true don't cares. The casez statement treats x as a don't care but demands that the z digit have an exact match.

```
module SevSegCase (aIn, sOut);
        input [3:0] aIn;
        output [6:0] sOut;
        reg [6:0] sOut;
        always @ (aIn)
                begin
                    case (aIn)
                         //                    abcdefg
                         4'b0000:sOut = 7'b0000001; //0
                         4'b0001:sOut = 7'b1001111; //1
                         4'b0010:sOut = 7'b0010010; //2
                         4'b0011:sOut = 7'b0000110; //3
                         4'b0100:sOut = 7'b1001100; //4
                         4'b0101:sOut = 7'b0100100; //5
                         4'b0110:sOut = 7'b0100000; //6
                         4'b0111:sOut = 7'b0001111; //7
                         4'b1000:sOut = 7'b0000000; //8
                         4'b1001:sOut = 7'b0001100; //9
                         4'b1010:sOut = 7'b0001000; //A
                         4'b1011:sOut = 7'b1000010; //B
                         4'b1100:sOut = 7'b0000111; //C
                         4'b1101:sOut = 7'b0000001; //D
                         4'b1110:sOut = 7'b0110000; //E
                         4'b1111:sOut = 7'b0000110; //F
                    endcase
                end
endmodule
```